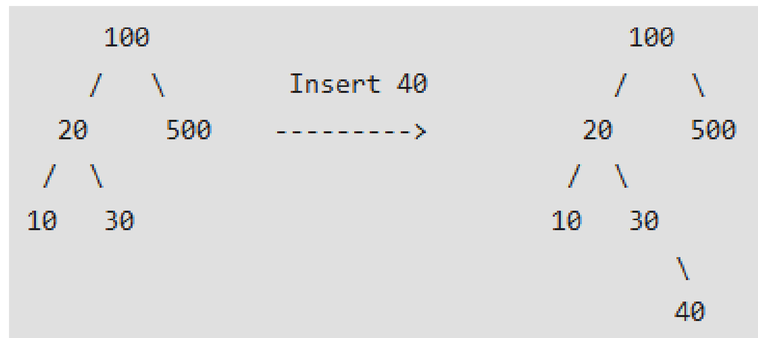
شجرة البحث الثنائية (Binary Search Tree)

هي الشجرة الثنائية التي تحقق التالي:

- في أي شجرة فرعية يسارية تكون قيم مفاتيح العقد أقل من قيمة مفتاح الجذر لها.
- في أي شجرة فرعية يمينية تكون قيم مفاتيح العقد أكبر من قيمة مفتاح الجذر لها.

إدخال قيمة Key في شجرة البحث الثنائية:

قيمة المفتاح الجديدة المراد إضافتها يجب دائماً أن تضاف كعقدة ورقة (Leaf) كما يوضح الشكل التالي عملية إضافة 40 كعقدة ورقة



مبدأ عمل الخوارزمية:

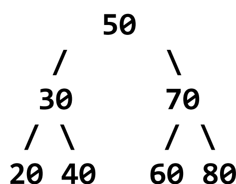
عند ادخال عقدة جديدة فستكون لدينا إحدى الحالات الثلاث التالية:

إذا كانت العقدة فارغة NULL: أنشئ عقدة جديدة وضع المفتاح المراد إدخاله فيها،

أما إذا كانت قيمة المفتاح المراد إدخاله أصغر من مفتاح الجذر لها: استدعي الادخال لليسار،

وإلا فقيمة المفتاح المراد إدخاله أكبر من مفتاح الجذر لها: استدعي الادخال لليمين.

مثال: اكتب كود خوارزمية إدخال (Insert) قيم شجرة البحث الثنائية التالية بلغة C++:



الحل:

ملاحظة: عند تطبيق التجوال inorder على شجرة بحث ثنائية فسيكون الخرج مرتب تصاعدياً.

```
// C++ program to demonstrate insertion
// in a BST recursively.
#include <iostream>
using namespace std;

class BST {
	int data;
	BST *left, *right;

public:
	// Default constructor.
	BST();

	// Parameterized constructor.
	BST(int);

	// Insert function.
	BST* Insert(BST*, int);

	// Inorder traversal.
	void Inorder(BST*);
};

// Default Constructor definition.
BST ::BST()
	: data(0)
	, left(NULL)
	, right(NULL)
{
}

// Parameterized Constructor definition.
BST ::BST(int value)
{
	data = value;
	left = right = NULL;
}

// Insert function definition.
BST* BST ::Insert(BST* root, int value)
{
	if (root==NULL) {
		// Insert the first node, if root is NULL.
		return new BST(value);
	}

	// Insert data.
```

```
	if (value > root->data) {
		// Insert right node data, if the 'value'
		// to be inserted is greater than 'root' node data.

		// Process right nodes.
		root->right = Insert(root->right, value);
	}
	else {
		// Insert left node data, if the 'value'
		// to be inserted is greater than 'root' node data.

		// Process left nodes.
		root->left = Insert(root->left, value);
	}

	// Return 'root' node, after insertion.
	return root;
}

// Inorder traversal function.
// This gives data in sorted order.
void BST ::Inorder(BST* root)
{
	if (!root) {
		return;
	}
	Inorder(root->left);
	cout << root->data << endl;
	Inorder(root->right);
}

// Driver code
int main()
{
	BST b, *root = NULL;
	root = b.Insert(root, 50);
	b.Insert(root, 30);
	b.Insert(root, 20);
	b.Insert(root, 40);
	b.Insert(root, 70);
	b.Insert(root, 60);
	b.Insert(root, 80);

	b.Inorder(root);
	system ("pause");
	return 0;
}
```